Bilkent University
CS-224 Lab 5
Preliminary Work

Full Name: Arda İynem
ID: 22002717
Department: Computer Science
Course: CS 224
Section: 1
Lab No: 5
Date: 30/11/2022

Part 1 (b)

$$RF[rd] \leftarrow RF[rd] + (RF[rs] >>> RF[rt])$$

Data Hazards

Compute-Use Hazard & Load-Use Hazard & Load-Store Hazard
When consecutive instructions operate on a common register, subsequent instructions may read the wrong data from the common register (destination register of first instruction) from the register file during Decode stage since the prior instruction’s Execute stage data is not written until the end of the write back stage or the Memory stage is not completed yet. The correct data value is not written in the source register of the next instruction which produces a compute-use hazard.

Affected Pipeline Stages
Subsequent instruction’s (which is after the computation instruction) **Decode** stage will get the wrong data value from the register file. Thus also remaining **Execute**, **Memory** and **WriteBack** stage will perform operations by using the wrong data value.

Control Hazards

Branch Hazard
Branch decision is not made by the time the next instruction is fetched from the instruction memory which causes a control hazard.

Affected Pipeline Stages
In subsequent instruction, **Fetch** stage will get the wrong address. Thus remaining **Decode**, **Execute**, **Memory** and **WriteBack** stage also will be affected by performing possibly wrong instruction.

Part 1 (c)

1) Compute-Use Hazard - Forwarding

Case 1: The correct value from the memory stage of the corresponding register is forwarded. If there is a one clock cycle difference between two instructions.

Case 2: If there is a two clock cycle difference between two instructions, value is forwarded from the beginning of the writeback stage.

2) Load-Use/Store Hazard - Stalling & Flushing & Forwarding

The instruction affected by this hazard is stalled in the decode stage. Pipeline between the decode and execute stages are flushed during the stall, in order to prevent writing meaningless values to registers. When the load word process of the previous instruction ends, the stalling also ends. After that, the correct register value can be forwarded from the beginning of the writeback stage.

3) Early Branch Data Hazard - Stalling & Flushing & Forwarding OR Forwarding

Case 1: If the correct value is the result of an ALU operation and it is in the memory stage, it can simply be forwarded.

Case 2: If it is in the execute stage, or if it is the result of a lw operation and it is in the memory stage, it can be stalled, flushed, and forwarded just like in case 1.

4) Branch Hazard - Branch Prediction & Flushing

The processor predicts that the branch will not be taken. If the branch is not taken, everything works correctly. If the branch is taken; the instructions that are fetched before the branch decision made are flushed.

Part 1 (d)

Logic for Forwarding

```
if ((rsE != 0) AND (rsE == WriteRegM) AND RegWriteM) then
      ForwardAE = 10
else if ((rsE != 0) AND (rsE == WriteRegW) AND RegWriteW) then
      ForwardAE = 01
else ForwardAE = 00

if ((rsE != 0) AND (rsE == WriteRegM) AND RegWriteM) then
      ForwardBE = 10
else if ((rsE != 0) AND (rsE == WriteRegW) AND RegWriteW) then
      ForwardBE = 01
else ForwardBE = 00


ForwardAD = (rsD != 0) AND (rsD == WriteRegM) AND RegWriteM
ForwardBD = (rtD != 0) AND (rtD == WriteRegM) AND RegWriteM
```

Logic for Stalling & Flushing

```
lwstall = ((rsD = = rtE) OR (rtD = = rtE)) AND MemtoRegE

StallF = StallD = FlushE = lwstall OR branchstall

branchstall =
BranchD AND RegWriteE AND (WriteRegE == rsD OR WriteRegE == rtD)
OR
BranchD AND MemtoRegM AND (WriteRegM == rsD OR WriteRegM == rtD)
```

Part 1 (e)

Hazards sracc Instruction May Cause:

1) **Control-Use (RAW Data) Hazard**

Solution: Forwarding

Since the values that will be written back to the register are already determined when the “Execute” stage ends, these values can be forwarded to the next instructions before the writeback stage ends. However *sracc* reads from three registers, hence the hazard elimination process should be done for three registers with the same logic that of two.

Case 1: One Clock Cycle Away Instruction (from sracc instruction)
Sracc will be at the Data Memory stage when this instruction is at the Execute stage. Therefore, values can be forwarded from the Data Memory stage of the sracc instruction.

Case 2: Two Clock Cycle Away Instruction (from sracc instruction)
Sracc will be at the beginning of the Writeback stage when this instruction is at the Execute stage. Therefore, values can be forwarded from the beginning of the Writeback stage of the sracc instruction.

Case 3: More Than Two Clock Cycle Away Instruction (from sracc)
No hazard will occur, since the values will have already been written back to the register file.

Test (With Control and Data Hazards)

```
# COMPUTE-USE HAZARD TEST
addi $s0, $zero, 3 #$s0 = 3
addi $s1, $zero, 47 #$s1 = 47
add $s2, $s0, $s1 #$s2 = 50
sracc $s0, $s2, $s0 #$s0 = 3 + (50 >> 3) = 9

# LOAD-USE & LOAD-STORE HAZARD TEST
sw $s0, 0x60($zero) # Memory 0x60 = 9
addi $s0, $s0, -7 #$s0 = 2
lw $s2, 0x60($zero) #$s2 = 9
sracc $s1, $s2, $s0 #$s1 = 47 + (9 >> 2) = 49
addi $s0, $s0, 47 #$s0 = 49

# EARLY BRANCH & BRANCH HAZARD TEST
beq $s0, $s1, -1 #infinite loop here
```

Test (With No Hazards)

```
addi $s2, $zero, 4
addi $s1, $zero, 7
addi $s2, $zero, 9
addi $s3, $zero, 3

# No Raw Data Hazards since commonly used registers
# Are away more than 2 cycles.
# Also no Load and Branch instructions exist
add $s4, $s1, $s2
sub $s5, $s2, $s1
sracc $s3, $s3, $s2
```